Co-Array Fortran
What is it? Why should you put it on BlueGene/L?

Robert W. Numrich

Minnesota Supercomputing Institute
University of Minnesota

rwn@msi.umn.edu

The Guiding Principle behind Co-Array Fortran

- What is the smallest change required to make Fortran 90 an effective parallel language?
- How can this change be expressed so that it is intuitive and natural for Fortran programmers?
- How can it be expressed so that existing compiler technology can implement it easily and efficiently?

What's the Problem with SPMD?

- One processor knows nothing about another's memory layout.
  - Local variables live on the local heap.
  - Addresses, sizes and shapes are different on different program images.
- How can we exchange data between such non-aligned variables?

Co-Array Fortran Extension

- Incorporate the SPMD Model into Fortran 90
  - Multiple images of the same program
  - Text and data are replicated in each image
- Mark some variables with co-dimensions
  - Co-dimensions behave like normal dimensions
  - Co-dimensions express a logical problem decomposition
  - One-sided data exchange between co-arrays using a Fortran-like syntax
- Require the underlying run-time system to map the logical problem decomposition onto specific hardware.

The CAF Execution Model

- The number of images is fixed and each image has its own index, retrievable at run-time:

  1 □ num_images()

  1 □ this_image() ≤ num_images()
- Each image executes the same program independently of the others.
- The programmer inserts explicit synchronization and branching as needed.
- An “object” has the same name in each image.
- Each image works on its own local data.
- An image moves remote data to local data through, and only through, explicit CAF syntax.

University of Minnesota

What is Co-Array Syntax?

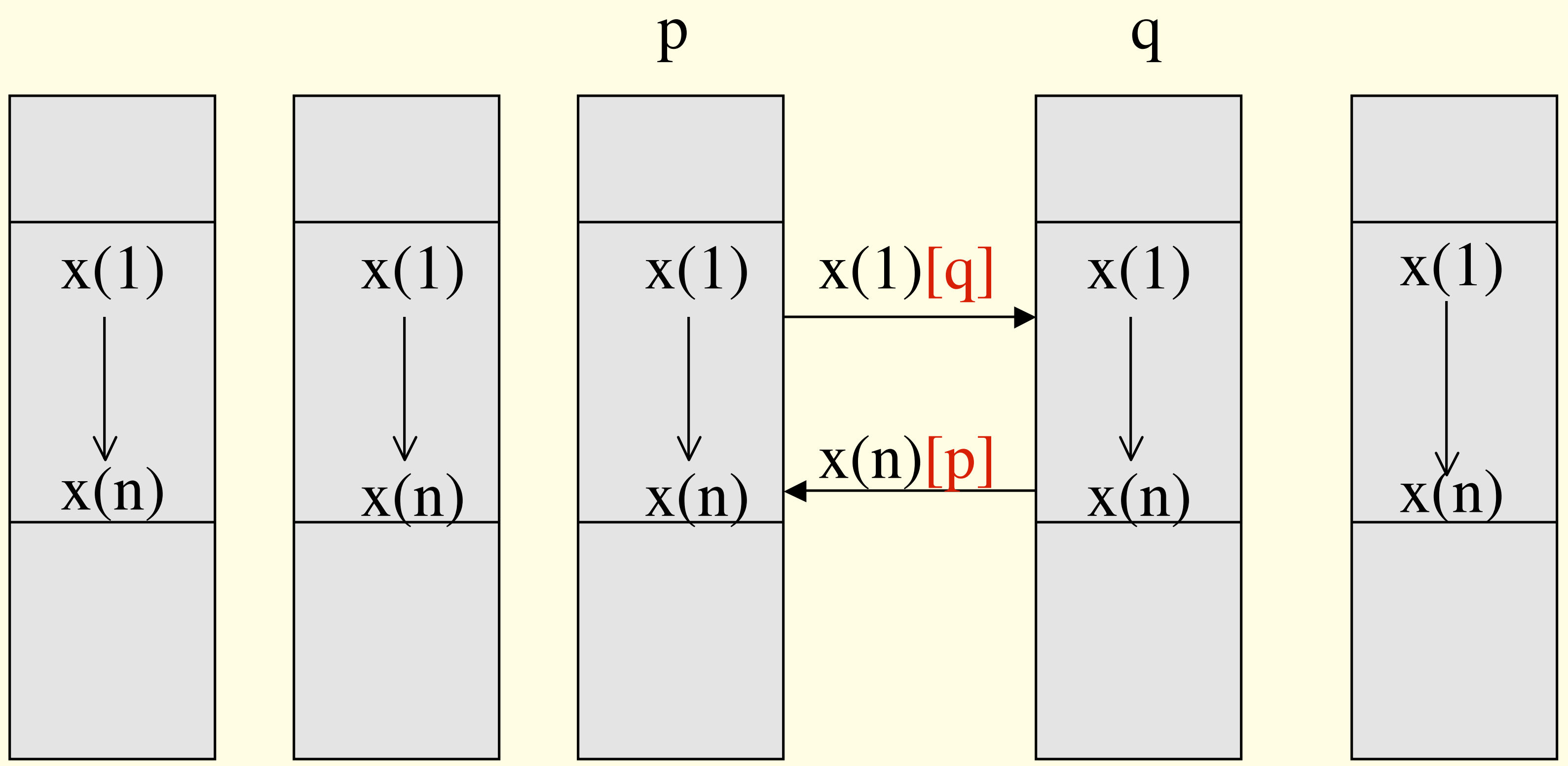
- Co-Array syntax is a simple extension to normal Fortran syntax.
  - It uses normal rounded brackets ( ) to point to data in local memory.
  - It uses square brackets [ ] to point to data in remote memory.
  - Syntactic and semantic rules apply separately but equally to ( ) and [ ].

Examples of Co-Array Declarations

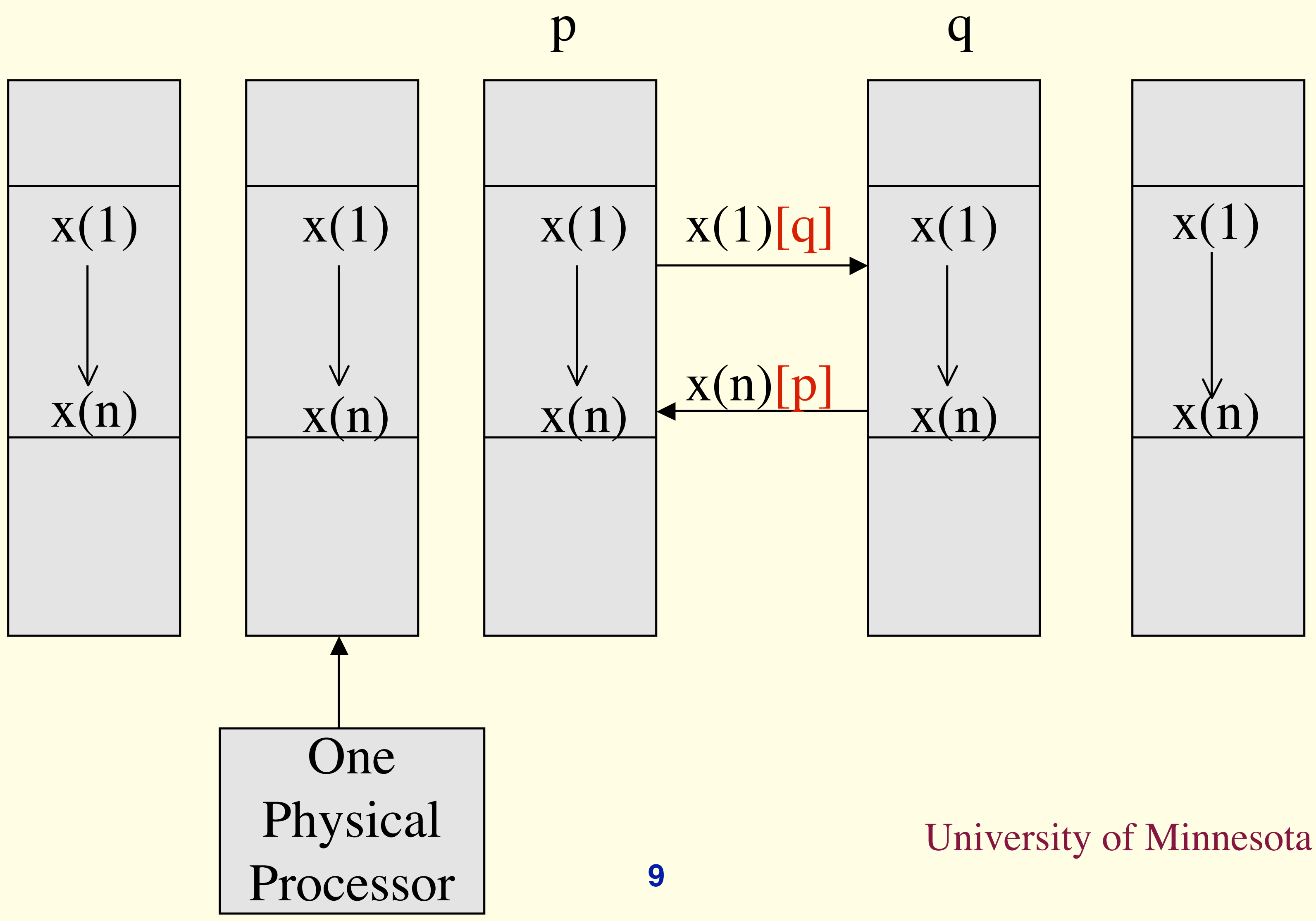
```
real :: s[*]
real :: a(n)[*]
complex :: z[*]
integer :: index(n)[*]
real :: b(n)[p, *]
real :: c(n,m)[0:p, -7:q, 11:*]
real, allocatable :: w(:)[:]
type(field) :: maxwell[p,*]
```

University of Minnesota

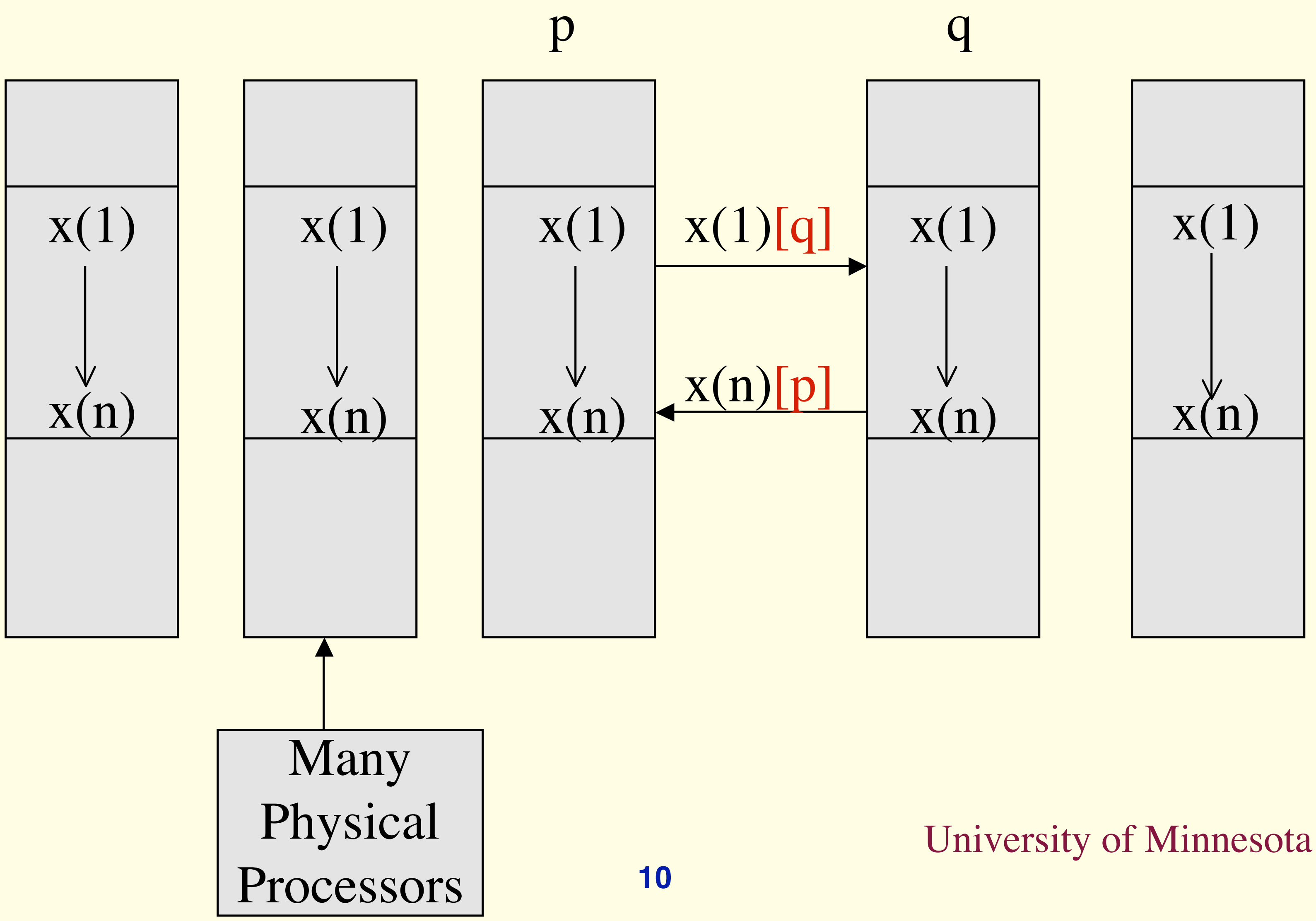
CAF Memory Model



One-to-One Execution Model

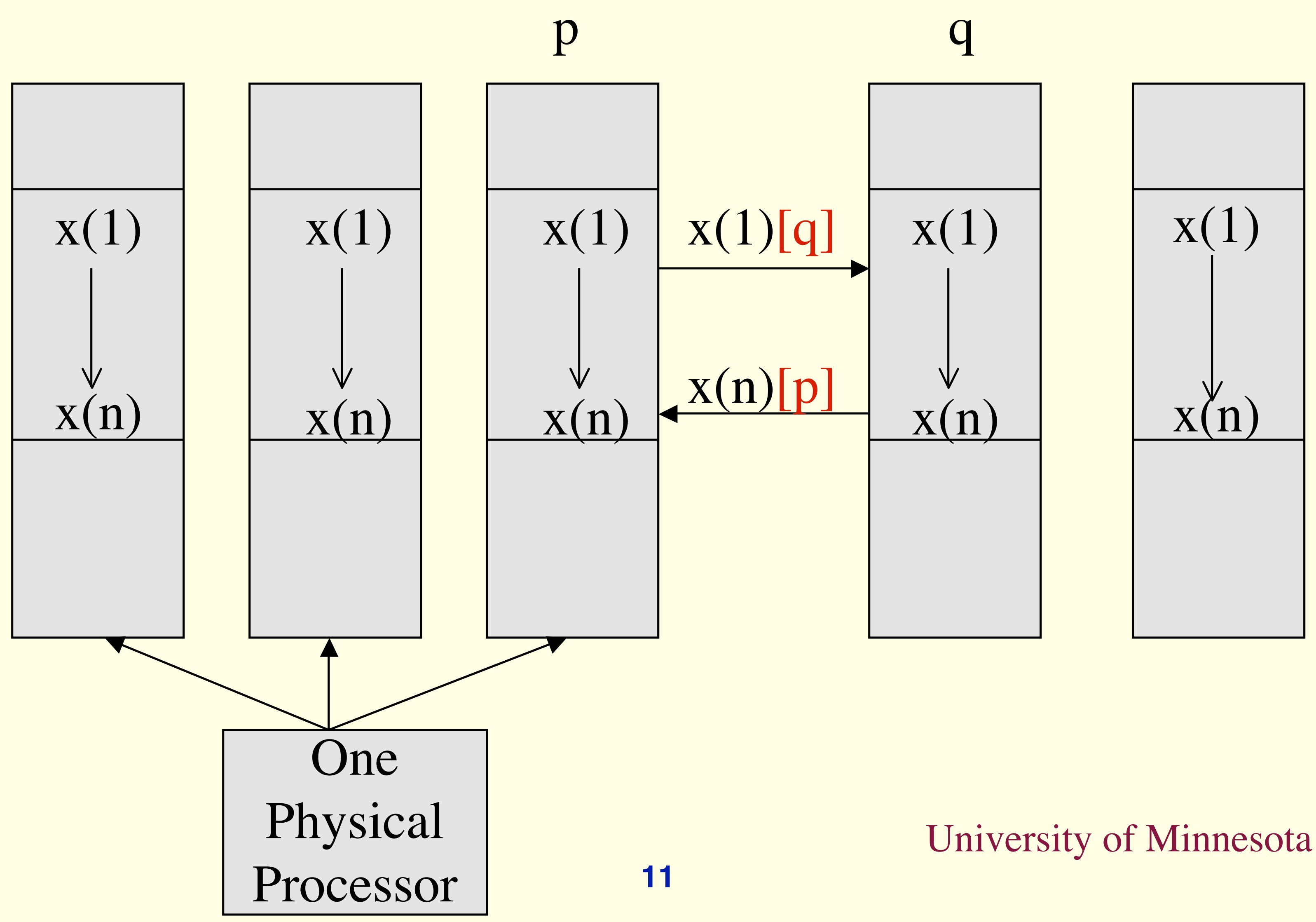


Many-to-One Execution Model



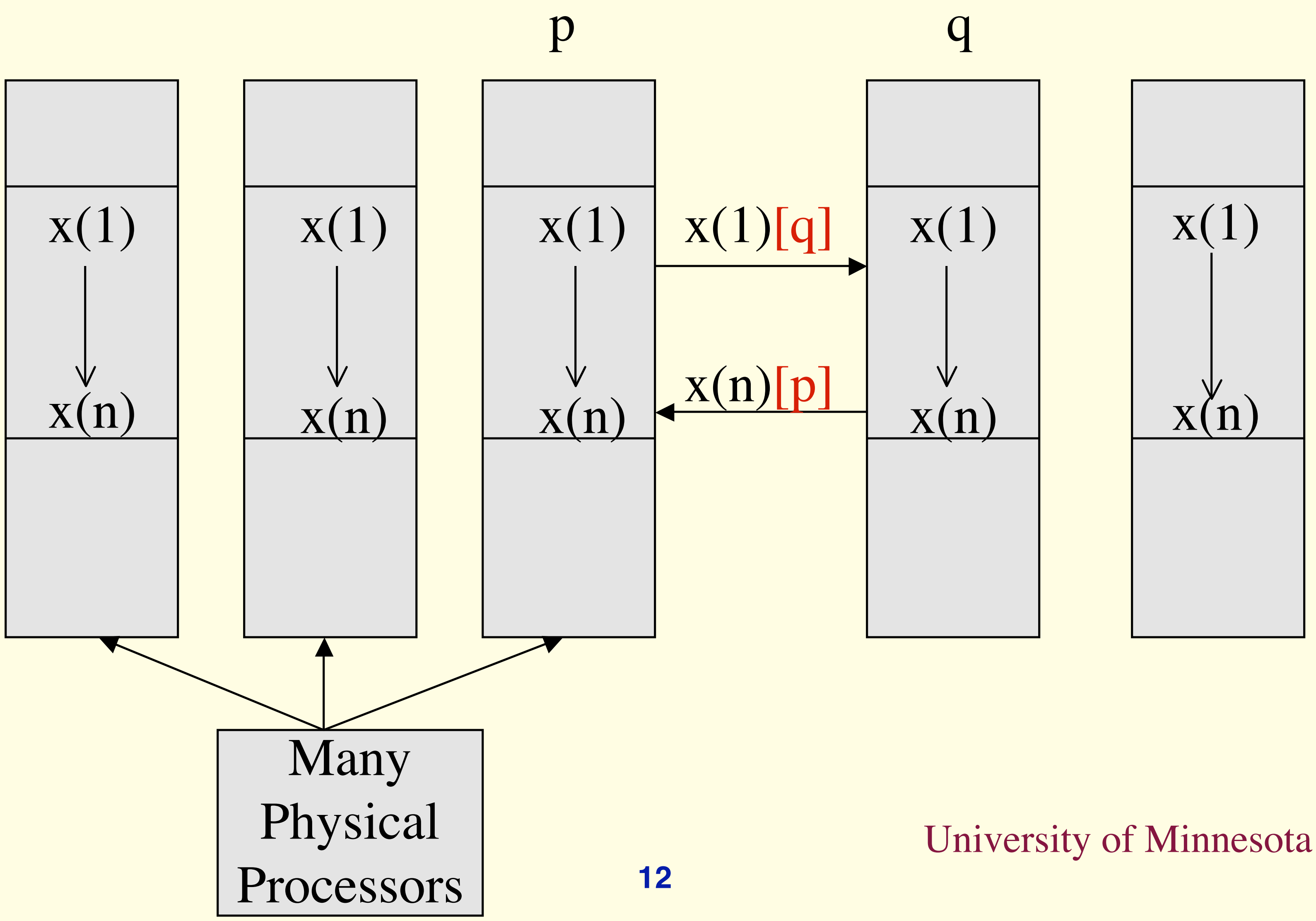
University of Minnesota

One-to-Many Execution Model



University of Minnesota

Many-to-Many Execution Model



University of Minnesota

What Do Co-Dimensions Mean?

real :: x(n)[p,q,□]

- Replicate an array of length n, one on each image.
- Build a map so each image knows how to find the array on any other image.
- Organize images in a logical (not physical) three dimensional grid.
- The last co-dimension acts like an assumed size array: □□ num_images()/(pxq)
- A specific implementation could choose to represent memory hierarchy through the co-dimensions.

University of Minnesota

Relative Image Indices (1)

	1	2	3	4
1	1	5	9	13
2	2	6	10	14
3	3	7	11	15
4	4	8	12	16

```
x[4,*]  this_image() = 15       this_image(x) = (/3,4/)
```

Relative Image Indices (II)

$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & 1 & 5 & 9 & 13 \\ 1 & 2 & 6 & 10 & 14 \\ 2 & 3 & 7 & 11 & 15 \\ 3 & 4 & 8 & 12 & 16 \end{array}$$

x[0:3,0:*]   this_image() = 15   this_image(x) = (/2,3/)

University of Minnesota

Relative Image Indices (III)

	0	1	2	3
-5	1	5	9	13
-4	2	6	10	14
-3	3	7	11	15
-2	4	8	12	16

x[-5:-2,0:*] this_image() = 15   this_image(x) = (/-3, 3/)

University of Minnesota

Relative Image Indices (IV)

	0	1	2	3	4	5	6	7
0	1	3	5	7	9	11	13	15
1	2	4	6	8	10	12	14	16

x[0:1,0:*]   this_image() = 15   this_image(x) =(/0,7/)

University of Minnesota

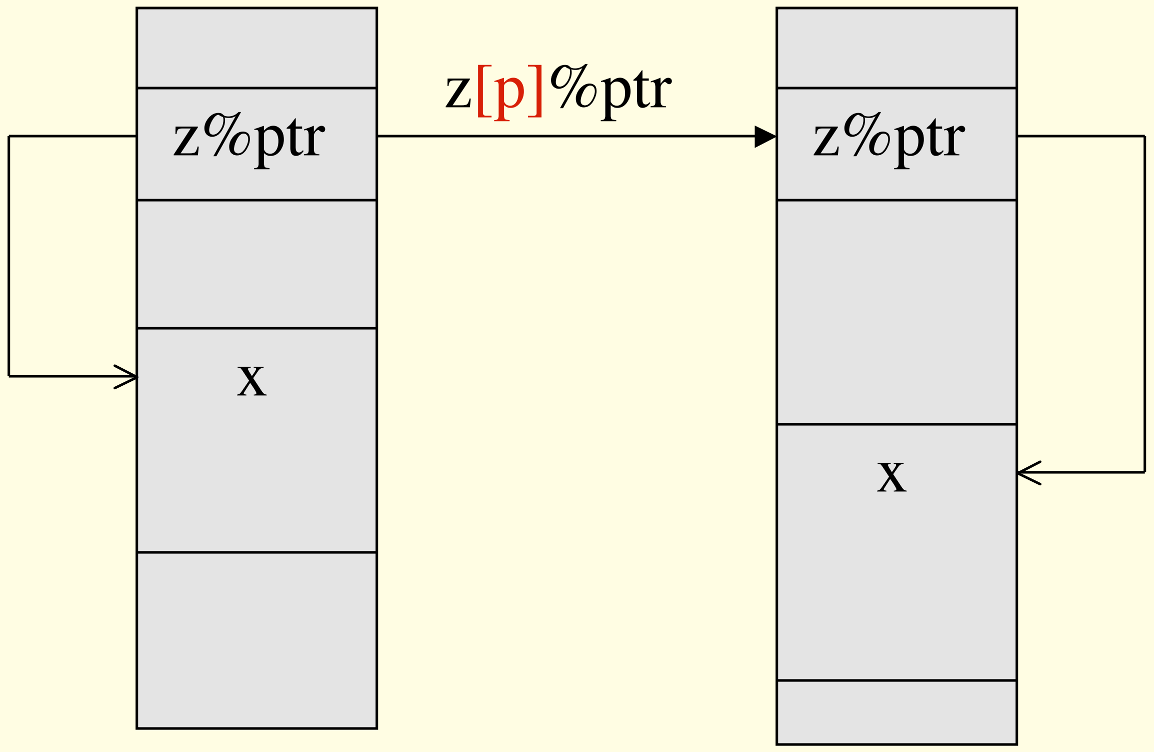
Communication Using CAF Syntax

```
y(:) = x(:)[p]

myIndex(:) = index(:)
yourIndex(:) = index(:)[you]

x(index(:)) = y[index(:)]

x(:)[q] = x(:) + x(:)[p]
```

Absent co-dimension defaults to the local object.

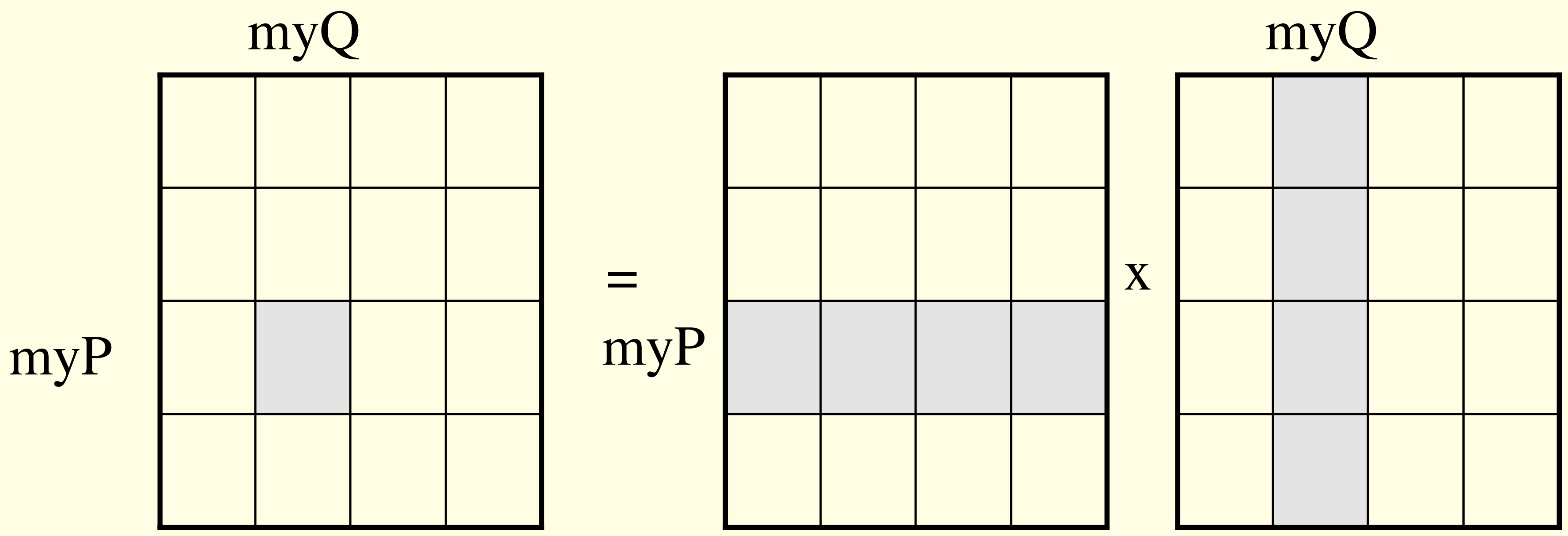
University of Minnesota

Irregular and Changing Data Structures



University of Minnesota

Matrix Multiplication



real,dimension(n,n)[p,□] :: a,b,c

(/myP,myQ/) = this_image(c)

University of Minnesota

Matrix Multiplication

```
real,dimension(n,n)[p,*] :: a,b,c

do k=1,n
 do q=1,p
    c(i,j)[myP,myQ] = c(i,j)[myP,myQ]
                                   + a(i,k)[myP, q]*b(k,j)[q,myQ]
 enddo
enddo
```

University of Minnesota

Matrix Multiplication

```
real,dimension(n,n)[p,*] :: a,b,c

do k=1,n
 do q=1,p
    c(i,j) = c(i,j) + a(i,k)[myP, q]*b(k,j)[q,myQ]
 enddo
enddo
```

Matrix Multiplication

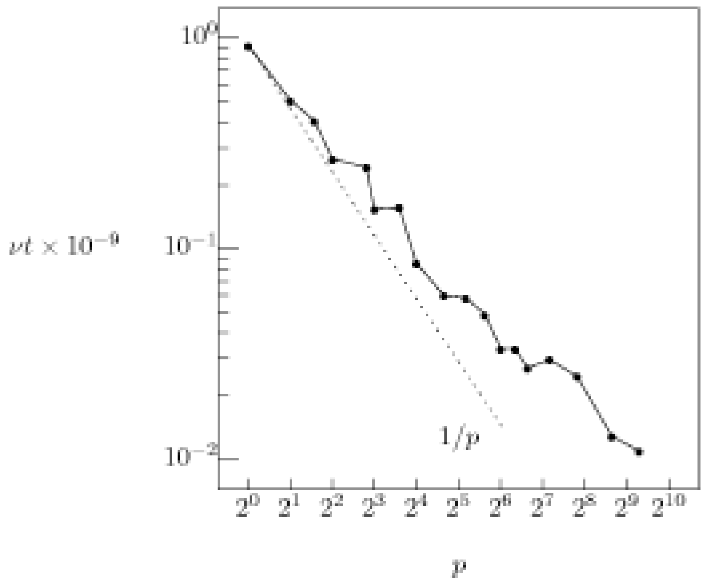


Figure 4: Time as a function of the number of processors $p = q \times r$ for block matrix multiplication. The matrix size is 1000×1000 with blocks of size $1000/q \times 1000/r$. Time is expressed in dimensionless giga-clock-ticks, $\nu t \times 10^{-9}$, as measured on a CRAY-T3E with frequency $\nu = 300$MHz. The dotted line represents perfect scaling.

Communication for LU Decomposition

- Row interchange
  - temp(:) = a(k,:)
  - a(k,:) = a(j,:) [p,myQ]
  - a(j,:) [p,myQ] = temp(:)
- Row “Broadcast”
  - L0(i:n,i) = a(i:,n,i) [p,p]   i=1,n
- Row/Column “Broadcast”
  - L1 (:,:) = a(:,:) [myP,p]
  - U1(:,:) = a(:,:) [p,myQ]

LU Decomposition

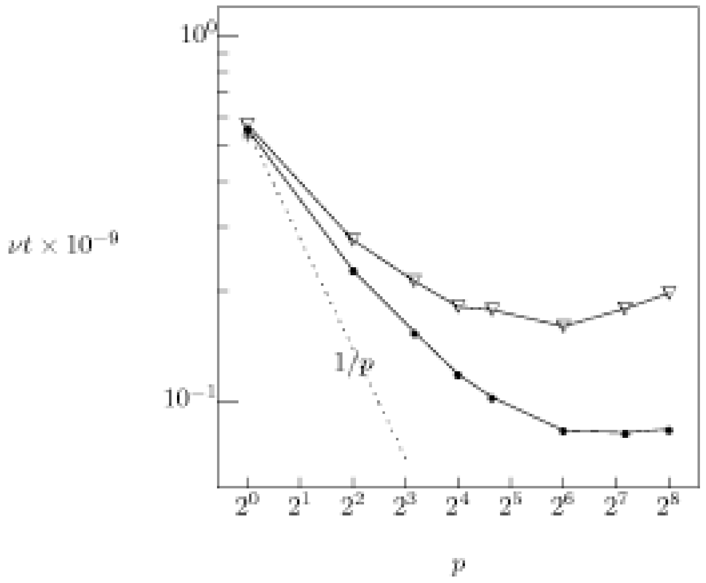


Figure 6: Time as a function of the number of processors $p = q \times r$ for block-cyclic LU decomposition. The matrix size is 1000×1000 with blocks of size 48×48. Time is expressed in dimensionless giga-clock-ticks, $\nu t \times 10^{-9}$, as measured on a CRAY-T3E with frequency $\nu = 300$MHz. The dotted line represents perfect scaling. The curve marked with bullets (•) is code written in Co-Array Fortran. The curve marked with triangles (▽) is SCALAPACK code.

A Parallel “Class Library” for CAF

- Combine the object-based features of Fortran 90 with co-array syntax to obtain an efficient parallel numerical class library that scales to large numbers of processors.
- Encapsulate all the hard stuff in modules using named objects, constructors,destructors, generic interfaces, dynamic memory management.
- Based on Vector Maps designed to support redistribution of data for load balancing, adaptive mesh refinement, etc.

Run-time System Support for CAF

- Compiler decodes CAF syntax and determines the processor (thread, process, node) where the data lives
- Compiler hands this information to a communication protocol
  - Global virtual address space: use load/store instructions
    - Higher-order bits in address: remote = local + shift(p)
    - Virtual offset: remote =local + offset(p)
    - Table lookup: remote = remote(p)
  - Implement on one BG/L compute node as proof-of-concept?
  - Interface to a one-sided communication library
    - Armci, Shmem, Lapi, Quadrics elan, Myrinet GM-2, MPI-2, Active messages
- Dynamic memory management for co-arrays
- Fast barriers
- Cache coherence (invalidate on sync?)
- Optimal logical to physical mapping (simulated annealing?)

The Co-Array Fortran Standard

- Co-Array Fortran is defined by:
  - R.W. Numrich and J.K. Reid, “Co-Array Fortran for Parallel Programming”, ACM Fortran Forum, 17(2):1-31, 1998
- Additional information on the web:
  - www.co-array.org
  - www.pmodels.org

Why Language Extensions?

- Programmer uses a familiar language.
- Syntax gives the programmer control and flexibility.
- Compiler concentrates on local code optimization.
- Compiler evolves as the hardware evolves.
  - Lowest latency and highest bandwidth allowed by the hardware
  - Data ends up in registers or cache not in memory
  - Arbitrary communication patterns
  - Communication along multiple channels